CS434 Final Project Report

Ciin Dim, Daniel Ross, Logan Wingard

6/12/2018

1 Feature formulation and preprocessing

1.1 Features

The features we fed to our learning algorithms were the original nine features given: Time, Glucose, Slope, IOB, MOB, Morning, Afternoon, Evening, and Night. Before using the data for training, we converted the time to seconds. We then reduced the original data by combining every 30-minute window into one average vector. Some vectors are an average of less than a 30-minute window due to gaps in time. We condensed our data in this way so that our algorithms could learn patterns based on a 30-minute time span instead of individual readings. Similarly, when reading in the test data, we created a nine-feature vector for each row of 63 features. The vector is an average of the features of each data point in the window of seven data points provided in a row of the test instances.

1.2 Preprocessing

In addition to the data formatting previously mentioned, we reduced noise in the data before feeding them to our learning algorithms. While looking through the training data, we noticed that there were non-binary values under binary features such as Morning, Afternoon, Evening, and Night. To cast this data back to binary values, we replaced the values with 1 if it was greater than 1, and 0 otherwise. Reducing this noise was necessary to prevent the noisy values from having a large impact on learning.

2 Learning algorithms

2.1 Algorithms explored

2.1.1 Linear Regression

Since we are looking for a correlation between features such as time of day, blood sugar level, etc. and whether a subject has a hypoglycemic episode,

linear regression will allow us to calculate a line that best fits the training data to predict the outcome based on the features.

2.1.2 K Nearest Neighbor

Similar to linear regression, the relationship of the features are telling of whether a subject will have a hypoglycemic episode. Therefore, we are using k nearest neighbor to find instances most similar to the test instances to predict the outcome.

2.1.3 Logistic Regression

Logistic Regression starts it's calculation with an empty array of weights corresponding to the effect that a feature has on the result. Over the each iteration the features in the data set are multiplied by the obectives difference from the keys in the dataset. These multiplied values are added to the gradient and then the gradient is multiplied by the eta before being subtracted from the weights. The resulting process will drive the weight towards the dataset's optimum value over time.

2.2 Final models

K Nearest Neighbor: k = 11 **Logistic Regression:** Eta = 0.005 Iterations = 800

3 Parameter Tuning and Model Selection

3.1 Linear Regression

About the only thing you can do to tune the parameters of linear regression would be to normalize the variables or not. While there is a normalize function that is just a sigmoid function, we do not use it as we feel the results not using it are more accurate than when we did use it.

3.2 K Nearest Neighbor

The parameter we tuned for our KNN model was k (the number of neighbors to compare a test point to). We chose the value k = 11 by using cross validation with the training sets. For each value of k from 1 to 20, we trained on most of the training data and used a 30-minute window validation set to get the results. We found that a value of k higher than 11 resulted in little improvement on error. k = 11 gave us the highest accuracy on average.

3.3 Logistic Regression

The two key parameters in Logistic Regression are the Learning Rate and the number of Iterations used in finding the gradient. We set up nested while loops to test Eta values between 0.005 and 0.1 in increments of 0.005 with every number of iterations between 100 and 1000 in increments of 100. This compared 200 weight matrices to see which was the most accurate when tested on the concatentation of Subject 1, Subject 4, Subject 6, and Subject 9. A key caveat was that if a weight returned more false negatives than there were true positives in the data the accuracy was set to 0

4 Results

To get results from our algorithms, we ran knn.py for method 1, lin.py for method 2, and logreg.py for method 3. The prediction files were output to the predictions/ folder. Since most of the data labels were 0, there were a lot more false negatives than false positives when training KNN. This is due to the fact that a majority of neighbors to any given point were labeled 0, pushing those point to that same label. The parameter k was tuned mainly to reduce false negatives by considering more neighbors.